FactoryMethodPattern

* 여기에서는 템플릿 메소드 패턴이 사용됨
* 구조와 구현의 분리

장점

1. 다른 값을 생성시 Creator 와 Item 을 구현시 다른 값을 만들 필요가 없음
2. 접근하는 클래스에서 변경 할 필요 없음

예시 소스

package designpattern;

import java.sql.Date;

public class FactoryMethodPattern {

public static void main(String[] args) {

// 요구사항 게임 아이템 과 생성을 구현 -> 아이템을 생성전 db에서 아이템정보를 요청

// 아이템 생성후 db에 아이템 생성정보를 남김

ItemCreator creator;

Item item;

creator = new HpCreator();

item = creator.create();

creator= new MpCreator();

creator.create();

}

}

abstract class ItemCreator{ //template method pattern 처럼 됨

public Item create(){

Item item;

requestItemInfo();

item = createItem();

createItemLog();

return item;

}

// 아이템 생성전 아이템 정보를 요청

abstract protected void requestItemInfo();

// 아이템 정보 남김

abstract protected void createItemLog();

//아이템을 생성하는 알고리즘

abstract protected Item createItem();

}

// 팩토리 메소드 -> 템플릿 메소드

interface Item{

public void use();

}

class HpPotion implements Item{

@Override

public void use() {

System.out.println("체력회복 ");

}

}

class MpPotion implements Item{

@Override

public void use() {

System.out.println("마력회복");

}

}

class HpCreator extends ItemCreator{

@Override

protected void requestItemInfo() {

System.out.println("db에서 체력 회복 물략의 정보를 가저옴 ");

}

@Override

protected void createItemLog() {

System.out.println("체력 회복 물약을 새로 생성 했습니다."+ new Date(0));

}

@Override

protected Item createItem() {

// 작업 ....

return new HpPotion();

}

}

class MpCreator extends ItemCreator{

@Override

protected void requestItemInfo() {

System.out.println("db에서 마력 회복 물략의 정보를 가저옴 ");

}

@Override

protected void createItemLog() {

System.out.println("마력 회복 물약을 새로 생성 했습니다."+ new Date(0));

}

@Override

protected Item createItem() {

// 작업 ....

return new MpPotion();

}

}